Designing Practical Polar Codes Using Simulation-Based Bit Selection

Shuanghong Sun, *Student Member, IEEE*, and Zhengya Zhang, *Member, IEEE*

***Abstract*— The frozen set selection of polar codes, known as bit selection, determines the error-correcting performance of polar codes. The original bit selection was derived for successive cancellation decoding in a binary erasure channel. Density evolution has been used to evaluate the bit error probability in binary memoryless channel, but the computational complexity is still high and the simplified versions rely on different degrees of approximations. We propose an alternative simulation-based in-order bit selection method that evaluates the error rate of each bit using Monte Carlo decoding simulations and selects the frozen set based on the bit reliability ranking. The simulation-based method does not rely on channel models and it can be applied to any practical channels in the field. The simulation can be accelerated on an FPGA platform to significantly shorten the time required to one day for a 1024-b code design. We use three examples to demonstrate the in-order bit selection method, a (256, 128) code, a (512, 256) code, and a (1024, 512) code. Compared with the codes designed using density evolution for an AWGN channel, our (256, 128) code shows a competitive BER; our (512, 256) code outperforms at low SNR; and our (1024, 512) code outperforms across a wide range of SNR by 0.3 to 0.6 dB. The algorithm and methodology are applicable to any code rate and longer code lengths.**

***Index Terms*— Polar code, bit selection, belief propagation decoding, decoder architecture, FPGA emulation.**

I. Introduction

NEWLY invented polar codes [1] have attracted much interest in 5G applications because of their promising capacity-achieving potential and efficient encoder and decoder implementation [2], [3]. Compared to the state-of-the-art turbo codes and LDPC codes, the factor graph of any length $N = 2^n$ polar code is predefined and the successive cancellation (SC) decoding is deterministic. However, the SC algorithm decodes bit by bit in a serial manner, so the latency of SC decoding is $\mathcal{O}(N)$ [4], [5]. Iterative belief propagation (BP) decoding has been proposed as an alternative to SC decoding [6], [7]. BP decoding uses a flooding schedule to allow N messages to be passed in parallel, thereby reducing the decoding latency to $\mathcal{O}(\log N)$. However, the error-correcting performance of BP decoding is worse than SC decoding [8].

Manuscript received February 06, 2017; revised May 24, 2017 and August 15, 2017; accepted September 7, 2017. Date of publication October 4, 2017; date of current version December 14, 2017. This work was supported in part by Intel Corporation and in part by NSF under Grant CCF-1054270. This paper was recommended by Guest Editor F. Sheikh. *(Corresponding author: Shuanghong Sun.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: shuangsh@umich.edu; zhengya@eecs.umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2017.2759253

Much efforts have been made to improve the error-correcting performance of polar codes, such as list SC decoding [9] that preserves a list of candidate decoding decisions, applying BP calculation in SC scheduling [10], and concatenation with outer codes [11]–[13]. Implementations based on the above algorithms [14], [15] have shown performance improvement of polar codes, at the cost of hardware requirements and/or design complexity.

When the block length of a polar code is sufficiently long, the capacity of the effective channel that each bit passes through polarizes to either almost 1 or almost 0 [1]. High-capacity reliable bits are to be used to carry information, and low-capacity unreliable bits are frozen to 0 to guarantee a good error rate. The bits that carry information are called information bits or non-frozen bits. The selection of the set of frozen bits is crucial to the error-correcting performance of polar codes. The code rate is adjusted by the size of the frozen set, without changing the codes' factor graph.

The selection of the frozen set is determined by the error probability or erasure probability of each bit. In SC decoding, the erasure probability of each bit can be derived and upper bounded for a binary erasure channel (BEC) [1]. Fig. 1 shows the capacities of the channel each bit passes through for a $N = 1024$ polar code, with erasure rate $\epsilon = 0.5$ in Fig. 1(a) and $\epsilon = 0.2$ in Fig. 1(b). Note that the channel contains an encoder, and the derivation assumes SC decoding [1]. The frozen set is chosen to be the set of bits with low-capacity channels. The choice of the frozen set depends on the communication channel. For a fixed code rate R_c, the frozen sets are different for channels of different statistics, as evidenced by the difference between Fig. 1(a) and Fig. 1(b). Moreover, the frozen set also depends on the decoding algorithm.

Arikan [1] used the Bhattacharyya parameter as an upper bound of the error probability of each bit in SC decoding. In a binary erasure channel (BEC), the Bhattacharyya parameter equals the erasure probability, and it can be efficiently evaluated with linear complexity. However, for an arbitrary binary memoryless channel, the complexity of the method becomes exponential in code length. Mori and Tanaka [16], [17] proposed density evolution to evaluate the bit error probability, as SC decoding of each bit can be modeled as BP decoding in a tree structure. However, Mori and Tanaka's method is also bottlenecked by high computational complexity due to high memory usage that grows exponentially with code length.

Tal and Vardy [18] extended the density evolution method by quantization to reduce the memory requirement. The

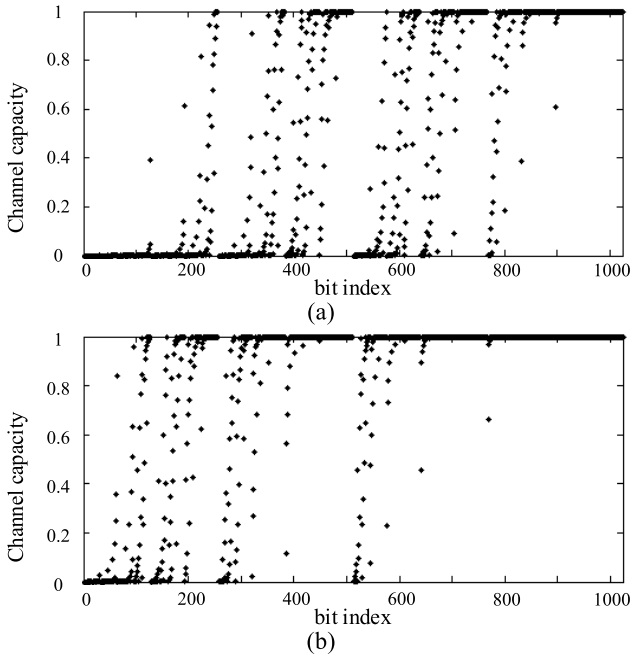


Fig. 1. Channel capacity of the 1024-bit polar code in BEC channel with erasure rate of (a) 0.5 and (b) 0.2.

method obtains a lower bound and an upper bound on the bit error probability given a specified maximum number of quantization levels. The number of quantization levels needs to be high to achieve a good accuracy. Alternatively, Trifonov [19] used Gaussian approximation in density evolution to reduce its computational complexity. These simplified density evolution methods offer substantial speedup and simplification of the bit error evaluation, but they also require approximations. Another drawback of these methods is that they require the channel model to be known in advance.

In this work, we propose a simulation approach to evaluate the error probability of each bit as an alternative method to density evolution. Inspired by Mori and Tanaka's formulation that views every step of SC decoding as BP decoding in a tree structure, we use Monte Carlo simulations of BP decoding to evaluate the error probability of each bit. Each simulation is exact and does not rely on any approximation. The simulation accounts for the finite code length, loops, numerical quantization, etc. The simulation method will be particularly useful in handling practical channels that sometimes have no closed-form mathematical representation. The bit error probabilities obtained from simulations account for practical non-idealities, including decoder implementation and its numerical precision.

For an N-bit polar code, the simulation-based bit selection method is done in N steps, where each step involves Monte Carlo simulations to measure the BER of one bit. At the end of N steps, a ranked BER list is produced. The simulation-based bit selection method works for all code rates by selecting the required number of bits to freeze from the list. Our method can also be extended to different code lengths. To facilitate the design of decoders for different code lengths, we have created a library and script approach, which requires minimal

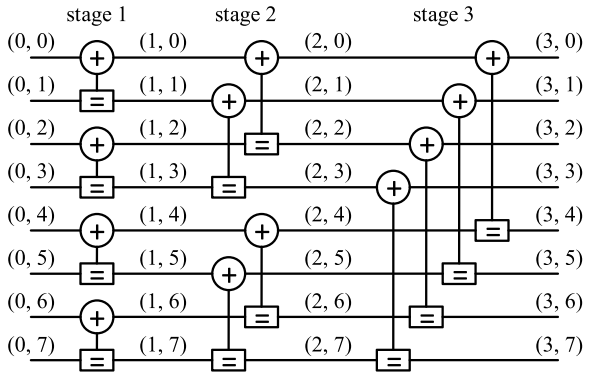


Fig. 2. Factor graph for encoding the 8-bit polar code.

effort to construct a decoder. We use FPGA to achieve significant accelerations. The bit selection process requires minimal supervision, and it can be done entirely autonomously. Compared to other published FPGA-based polar decoder emulators [20], [21], our platform is used specifically for bit selection. In addition to accelerating polar decoding, a software loop around the FPGA accelerator was added to set up the frozen patterns and collect the appropriate BERs. We have designed new approaches to speed up Monte Carlo simulations to make practical bit selections feasible. Although we use FPGA in this work, the simulation-based bit selection method can be programmed on a GPU or CPU cluster to achieve acceleration.

As a proof-of-concept, we demonstrate the simulation-based bit selection for three polar codes of block lengths of 256 bits, 512 bits, and 1024 bits. The results show up to 0.6 dB improvement in SNR (E_b/N_0) in BP decoding over the well-known bit selection obtained by density evolution [18].

II. Background

A polar code has a length of $N = 2^n$, and the code rate $R_c = K/N$ can be anywhere between 0 and 1. The generator matrix G is the n-th Kronecker power of matrix $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, i.e., $G_N = F^{\otimes n}$ with size $N \times N$.

The factor graph corresponding to the G matrix is shown in Fig. 2. The basic node consists of a plus sign and an equal sign, where the plus sign represents modulo-2 addition and the equal sign represents pass through. The node essentially implements multiplication of a 2-bit input vector by matrix F. There are $\log N$ stages of nodes and each stage consists of $N/2$ nodes. To perform polar encoding, an N-bit input message $\mathbf{u}$ is passed from the left hand side of the factor graph, and an N-bit codeword $\mathbf{x}$ is obtained from the right hand side.

Among the N message bits, there are K information bits and $N - K$ frozen bits. The information set $\mathcal{A}$ is defined as the index set of the information bits, and the frozen set $\mathcal{A}_c$ is defined as the index set of the frozen bits. Bit u_i is a frozen bit if $i \in \mathcal{A}_c$.

Used in a typical communication system, the codeword $\mathbf{x}$ is modulated and sent over a communication channel. The channel injects noise to the codeword, and produces noisy

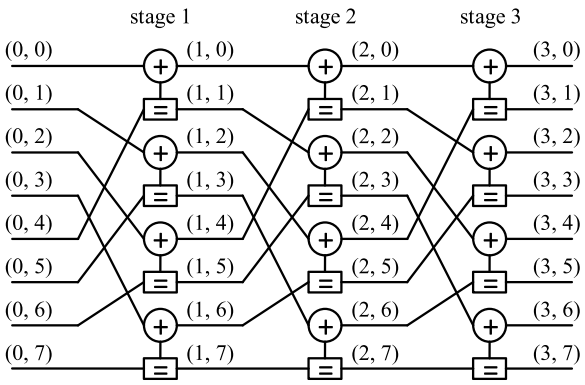


Fig. 3. Permuted factor graph of the 8-bit polar code.

codeword $\mathbf{y}$ as the output. A polar decoder will attempt to recover $\mathbf{u}$ from $\mathbf{y}$. The decoding can be visualized using the same factor graph shown in Fig. 2, except that the input $\mathbf{y}$ is provided from the right hand side, and the decoded codeword $\hat{\boldsymbol{u}}$ is obtained from the left hand side. In the following, we briefly review BP decoding of polar codes, as it is central to our discussions.

The BP algorithm decodes bits u_0 to u_{N-1} in parallel. BP decoding works by passing the frozen set information from left to right (in R propagation or simply R-prop) and passing the channel output $\mathbf{y}$ from right to left (in L propagation or simply L-prop) following the factor graph. One R-prop and one L-prop constitute a decoding iteration. Convergence can usually be reached in a few iterations. It is customary to permute the original factor graph in Fig. 2 to the form shown in Fig. 3 in a bit-reversal manner [6], so that the wiring between stages are kept the same to simplify a time-multiplexed implementation.

Note that although SC and BP decoding work on the same factor graph, the major difference between the two is that BP does not impose a sequential order of decoding, and the messages are "flooded" across the factor graph.

The basic node used in BP decoding is shown in Fig. 4 [6], where i is the bit index and j is the stage index. The left-bound messages (L messages) and right-bound messages (R messages) are calculated by

$$\begin{aligned} L_{j,i} &= f(L_{j+1,2i+1} + R_{j,i+N/2},\ L_{j+1,2i}) \\ L_{j,i+N/2} &= f(L_{j+1,2i}, R_{j,i}) + L_{j+1,2i+1} \\ R_{j+1,2i} &= f(L_{j+1,2i+1} + R_{j,i+N/2}, R_{j,i}) \\ R_{j+1,2i+1} &= f(L_{j+1,2i}, R_{j,i}) + R_{j,i+N/2} \end{aligned}$$

where the f function is identical to the f function used in SC decoding. $f(a, b) \approx \text{sign}(a)\text{sign}(b)\min\ (|a|, |b|)$.

A decoding iteration starts with R-prop from stage 1 to stage $\log N - 1$ to propagate frozen set information, followed by L-prop from stage $\log N$ to stage 1 to propagate channel outputs. Although Fig. 4 indicates that a node computes four output messages at a time – two L messages and two R messages, the calculation can be made uni-directional at any given time. That is, in an R-prop, a node computes only two R messages; and in an L-prop, a node computes only two L messages. In addition, the calculations in R-prop and L-prop are identical, enabling the same hardware node to be used in both R-prop and L-prop.

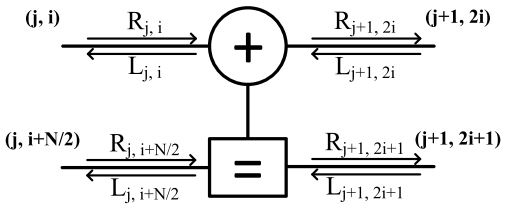


Fig. 4. Basic node of a BP decoder.

At the end of each iteration, the $L_{0,i}$ messages produced by stage 1 in the L-prop are taken as the soft decisions. The signs of the soft decisions are the hard decisions. The FER and BER improve with more iterations. The decoding latency is $n_{it}(2\log N - 1)$, assuming each stage being processed in parallel and n_{it} iterations are performed.

III. Decoder Architecture and Fast Design Methodology

Our bit selection methods rely on Monte Carlo decoding simulations, which can be highly time consuming. In order to make the bit selection methods practical, it is necessary to use an accelerator, e.g., an FPGA, to speed up the simulations, and automate the mapping of the decoder on the accelerator to reduce the design effort. Before we introduce the bit selection methods, we will briefly discuss our FPGA accelerator design and the mapping procedure to facilitate the bit selection experiments.

A. Decoder Architecture

The factor graph of a polar code can be mapped to a fully parallel architecture with each node mapped to a processing element (PE) and edges mapped to wires. For an N-bit polar code, a fully parallel architecture requires $\frac{N}{2}\log N$ PEs and $NQ\log N$ wires for connecting the PEs (assuming the message bit width is Q). The fully parallel architecture offers the highest throughput, but the large number of PEs coupled with numerous wires complicate the design, making it less scalable.

The natural way to partition the fully parallel architecture is along the stage boundaries. Such a partition results in a stage-parallel architecture that utilizes only one column of $\frac{N}{2}$ PEs for a N-bit polar code. The column of PEs will be time-multiplexed between $\log N$ stages, sacrificing throughput by a factor of $\log N$ but reducing the implementation complexity by approximately the same factor. The reduction in complexity is an important consideration as it ensures that a decoder for a sufficiently long code can be mapped to widely available FPGA platforms, and a lower complexity translates to faster hardware synthesis, placement and routing. The reduction in throughput can be recouped by using parallel hardware modules.

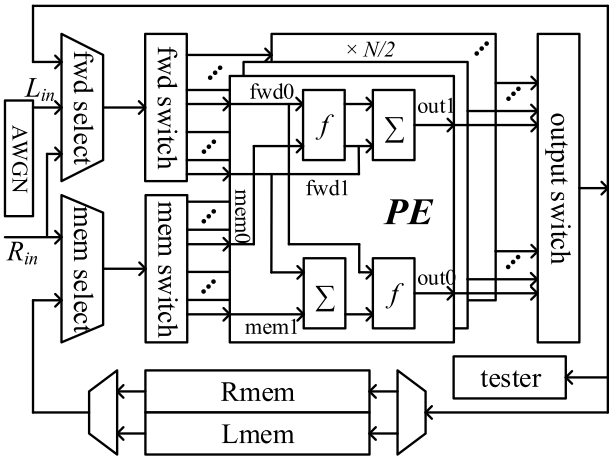


Fig. 5. FPGA block diagram.

In the stage-parallel architecture, a PE consists of two f functions (compare-select and XORs) and two adders to implement message processing. With bit-reversal shuffling, the wiring pattern between stages can be made the same, but switches are still needed to enable the use of the same PEs for both R-prop and L-prop in message-passing decoding. Specifically, three sets of switches are required as shown in Fig. 5: output switches, forwarding switches, and memory switches. The switches are implemented in NQ 2-to-1 MUXs to choose between R-prop and L-prop. The PE input selections are used to choose the appropriate inputs to the PEs for different stages. The forwarding selections are implemented in NQ 3-to-1 MUXs to select among forwarding, loading test vectors at the start of L-prop, and loading frozen set information at the start of decoding. The memory selections are implemented in NQ 2-to-1 MUXs to select between memory read and loading frozen set information at the start of decoding.

Two memories, Lmem and Rmem are used to store L messages and R messages. The column of $\frac{N}{2}$ PEs read/write NQ bits from/to the memory in parallel. Each memory word is NQ-bit wide, and each memory stores $\log N - 1$ words. In an R-prop, L messages are read from Lmem, and new R messages are stored in Rmem. Similarly in an L-prop, R messages are read from Rmem, and new L messages are stored in Lmem. The switching between the two memories is implemented by NQ 2-to-1 MUXs and DEMUXs.

To perform real-time emulation, an AWGN channel emulator and a built-in tester are integrated with the decoder to provide test vectors and to collect decoding errors. Our AWGN channel emulator was based on AWGN noise generators implemented using Box-Muller Transform. These AWGN noise generators can be conveniently instantiated through Xilinx LogiCORE. The AWGN noise is scaled according to the given channel SNR and added to BPSK-modulated bits in forming the input vectors for the decoder.

The decoder takes the frozen set information as R inputs (R_{in}) and the channel outputs as L inputs (L_{in}). R_{in} is set to 0 if a bit is an information bit, and R_{in} is set to the maximum allowed value if a bit is a frozen bit.

	cycle	1	2	3	4	…	n-1	n	n+1	n+2	n+3	n+4	…	2n-2	2n-1	2n	2n+1
read	L/Rmem	Lmem							Rmem							*Rin*	
	address	1	2	3	4	…	n-1		n-1	n-2	n-3	n-4	…	2	1		
proc.	L/R-prop		R-prop						L-prop								
	stage		1	2	3	…	n-2	n-1	n	n-1	n-2	…	4	3	2	1	
write	L/Rmem		Rmem						Lmem								
	address		1	2	…	n-3	n-2	n-1	n-1	n-2	…	4	3	2	1		

Fig. 6. FPGA scheduling.

TABLE I
HARDWARE UTILIZATION OF STAGE-PARALLEL BP DECODERS ON A XILINX VIRTEX-6 SX475T FPGA

N	256	512	1024	total available resources
Reg	13k	20k	36k	595k
LUT	22k	44k	86k	298k
Slice	6k	15k	35k	74k
RAMB36E1	78	122	216	2128
DSP48E1	12	18	33	2016
Frequency	100MHz	90MHz	70MHz	

Lmem and Rmem entries are all initialized to 0. The stage-parallel decoder follows a read-process-write 3-stage pipeline and the schedule is shown in Fig. 6. One decoding iteration consists of $2 \log N$ cycles including $\log N - 1$ cycles of R-prop followed by $\log N$ cycles of L-prop and one pipeline stall in between. Note that cycle 1 and cycle $2n + 1$ represent the same cycle in Fig. 6, but they are separated in the figure for illustration.

B. Design Methodology

The decoder is built using a semi-automated method consisting of a module library and an assembly script to facilitate its reuse. The module library is made up of parameterized common blocks required for a decoder. A script is used to assemble blocks and set parameters. The library and script method allows one to easily construct different decoders in minutes, minimizing the hardware design effort.

In order to support different experiments without having to redesign the hardware, the decoder also incorporates run-time tunable parameters, including decoding iterations, algorithmic knobs, and the channel SNRs. These parameters are inputs of the FPGA at run time.

The FPGA resource utilization is listed in TABLE I for N-bit stage-parallel decoders (N = 256, 512, 1024). Even the largest design listed in the table consumes only a small fraction of the available resources on a Xilinx Virtex-6 SX475T FPGA. There is ample room to support decoders for even longer codes for practical applications, where the block lengths are usually limited to a few Kb.

The decoder is mapped to FPGA to accelerate the bit selection of polar codes. The frozen set is fed to FPGA as an N-bit input pattern, indicating whether each bit is frozen or not, and the decoder hardware design is independent of the frozen set. In this work, we are interested in how each

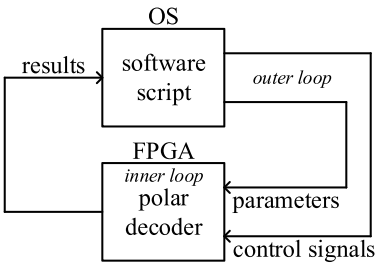


Fig. 7. Test setup block diagram.

frozen set affects the error-correcting performance of the code, the results of which guides the fine-tuning of the bit selection. Fig. 7 shows the test setup for bit selection: the inner loop of the bit selection is done on FPGA that runs Monte Carlo decoding simulations, and the outer loop is done in software that sets up the Monte Carlo simulations by sampling results from the decoder and setting run-time parameters and control signals.

IV. Simple Bit Selection Methods

We first develop two simple bit selection methods and discuss their weaknesses. The simple methods are developed for BP decoding using 256-bit polar codes. The results are applicable to codes of longer block lengths.

A. One-Time Rank-and-Freeze

The bit selection should be based on the error probability of each bit – the most reliable bits are used as information bits, and the least reliable ones are frozen. To measure the error probability of each bit of an N-bit polar code, we start with the rate-1 code using the method below.

1) Set all bits as information bits.
2) Run Monte Carlo BP decoding simulations and measure the error rate of each bit.
3) Rank the bits based on error rate and freeze the $N-K$ least reliable bits to obtain the bit selection for an (N, K) polar code.

We call this bit selection method one-time rank-and-freeze. Step 2 of this algorithm dominates the overall compute time, and we use FPGA to accelerate the Monte Carlo decoding simulations. The complexity of this bit selection method is $\mathcal{O}(N_{MC})$, where N_{MC} is the number of Monte Carlo decoding simulations.

Using a 6-bit fixed-point BP decoding in an AWGN channel (at an SNR of 7 dB), the measured BER of each bit of the (256, 256) polar code is shown in Fig. 8(a). The BER spreads over one order of magnitude. Based on the BER ranking, we select a rate-0.5 (256, 128) code with 128 bits of the worst BER frozen. The BER of each information bit of the rate-0.5 (256, 128) code spreads over two orders of magnitude, and improves by more than three orders of magnitude over the rate-1 code, as shown in Fig. 8(b). The difference between Fig. 8(a) and Fig. 8(b) shows the effect of freezing low-capacity bits: by freezing high-error bits, the performance of the remaining bits can be significantly improved.

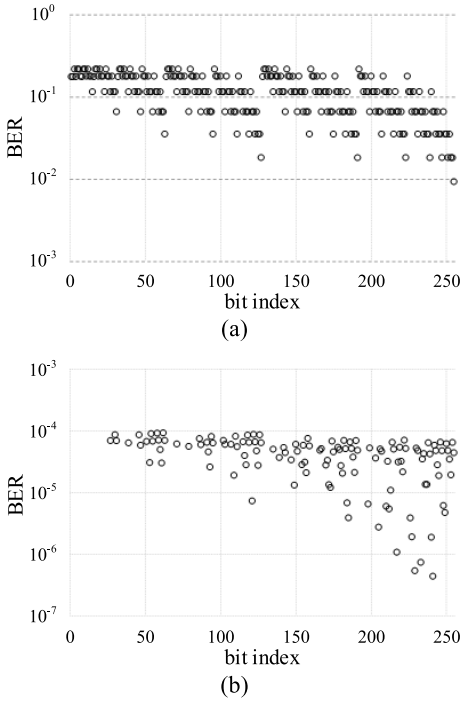


Fig. 8. BER of each bit of (a) a (256,256) polar code and (b) a (256,128) polar code.

However, at a relatively high SNR of 7 dB, a BER of nearly 10^{-4} is far from being satisfactory for a rate-0.5 code. Experiments at higher or lower SNR show no obvious improvement. The simple one-time rank-and-freeze method does not work well because it violates the precondition of channel polarization. The derivation of channel polarization is based on the precondition that when decoding bit i, all the former bits from 0 to $i-1$ are already known [1]. The one-time rank-and-freeze method evaluates the BER of bit i, while allowing all the remaining bits to be non-frozen. The flooding of messages back-and-forth over the factor graph allows low-capacity bits to affect the decoding of high-capacity bits. As a result, a high BER measured using this method does not necessarily indicate a low-capacity bit; and similarly, a low BER does not necessarily indicate a high-capacity bit either. Due to the unreliable BER measurement, this simple bit selection method is unsatisfactory.

B. Iterative Rank-and-Freeze

To account for the inter-bit dependence, the one-time rank-and-freeze method is refined to an iterative rank-and-freeze method. The idea is that instead of ranking all bits and freezing $N-K$ bits at one time, a part of $N-K$ bits are frozen at a time. After a part is frozen, the remaining non-frozen bits are evaluated and ranked again, based on which the next part of the frozen bits are chosen, until the desired code rate is obtained. The method is described below, where N_{it} is the number of iterations to be used.

1) Set all bits as information bits.
2) For $i = 1$ to N_{it}
   a) Run Monte Carlo BP decoding simulations and measure the error rate of each non-frozen bit.

b) Rank the non-frozen bits based on error rate and freeze the M_i least reliable bits.

Note that the number of bits to freeze in iteration i, namely M_i, is chosen such that $\sum_{i=1}^{N_{it}} M_i = N - K$. If smaller M_i values are chosen, more iterations are needed. The complexity of the iterative rank-and-freeze algorithm is $\mathcal{O}(N_{it} N_{MC})$. We use FPGA to accelerate the inner loop (Monte Carlo simulation), and the outer loop (iteration) is done using a script that interacts with the FPGA accelerator.

For ease of illustration, the BER of each non-frozen bit of a 256-bit polar code is sorted and displayed in a distribution shown in Fig. 9. Fig. 9(a) and Fig. 9(b) are two examples of the outcomes of running four iterations of iterative rank-and-freeze algorithm following slightly different procedures. In the first example shown in Fig. 9(a), the numbers of bits frozen in each iteration are {65, 98, 114, 122}, resulting in a (256, 191) code, a (256, 158) code, a (256, 142) code, and a (256, 134) code after the first, second, third and final iteration. In the second example shown in Fig. 9(b), the numbers of bits frozen in each iteration are {36, 67, 92, 112}, resulting in a (256, 220) code, a (256, 189) code, a (256, 164) code, and a (256, 144) code. It is evident from both examples that the impact of frozen set on BER is significant: after a few unreliable bits are frozen, the BER of the remaining bits are enhanced. As expected, the refined method produces better bit selections. However, the choices of the number of iterations and the number of bits to freeze in each iteration play important roles.

A closer look at the results unveils more insights. First, the bit selection is different depending on how the iterative procedure is carried out and how many bits are frozen in every iteration. For example, the BER of the (256, 142) code in the first example is higher than the BER of the (256, 144) code in the second example, although the former is a lower rate code. Second, the BER of the bit selection does not improve in a monotonic fashion with more iterations. For example, in Fig. 9(a), the BER of the (256, 158) code produced in the second iteration is not uniformly better than the BER of the (256, 191) code produced in the first iteration, although the former is derived from the latter by freezing some of the latter's non-frozen bits.

The iterative rank-and-freeze method improves upon the one-time rank-and-freeze method by freezing a portion of the bits at a time when evaluating BERs. In each iteration, the bits of the worst BER are frozen. In the next iteration, these bits will no longer affect decoding. How well the iterative rank-and-freeze method works depends on how many of the bits of the worst BER are frozen in each iteration. Fig. 9(a) and Fig. 9(b) illustrate two different outcomes depending on the number of bits frozen in each iteration. We note that the iterative rank-and-freeze method still violates the precondition of channel polarization. Therefore, the iterative rank-and-freeze method is still unsatisfactory.

V. In-Order Bit Selection Algorithm

In deriving channel polarization, SC decoding was used to decode polar codes in order, i.e., from u_0 to u_{N-1} [1].

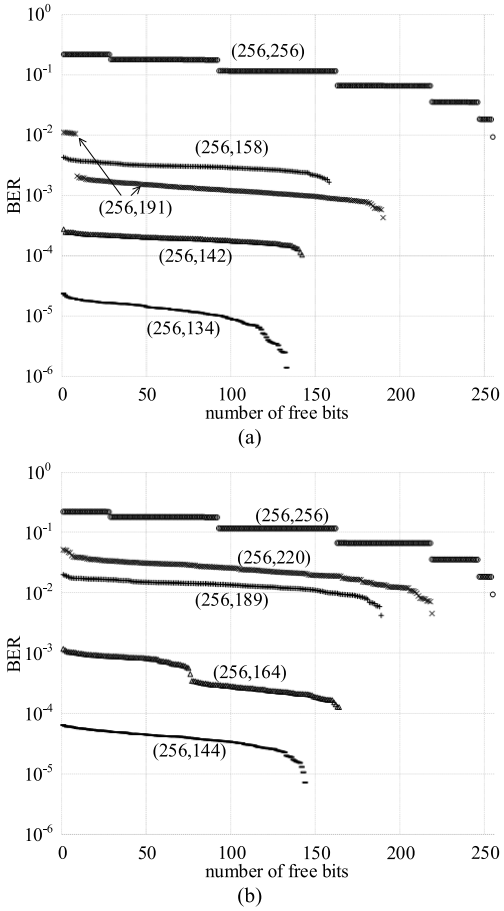


Fig. 9. Distribution of BER of each bit of two 256-bit polar codes using iterative rank-and-freeze algorithm.

u_0 is decoded first given channel outputs; next, given u_0 and channel outputs, u_1 is decoded; next, given u_0^1 (represents bits u_0 to u_1) and channel outputs, u_2 is decoded, and so on. The SC decoding of u_i depends only on the previously decoded bits u_0^{i-1} and channel outputs. If u_0^{i-1} are frozen, decoding of $\hat{u}_0^{i-1}$ is guaranteed to be correct and therefore $\hat{u}_i$ depends only on channel outputs, and the capacity of u_i can be accurately measured by the error probability.

Similarly in BP decoding, as an approximation of SC, if bits u_0^{i-1} are frozen and bits u_{i+1}^{N-1} are non-frozen, the error probability of u_i can be accurately measured. The data dependency under such condition in BP is the same as SC. The error probability measurement allows us to properly rank the bits and perform bit selection. This in-order bit selection method is elaborated below.

1) For $i = 0$ to $N - 1$
   a) If $i = 0$, then set all bits as information bits. If $i \geq 1$, then freeze bits u_0^{i-1} and set bits u_i^{N-1} non-frozen.
   b) Run Monte Carlo BP decoding simulations and measure the error rate of u_i.
2) Rank the bits based on error rate and freeze the $N - K$ least reliable bits to obtain the bit selection for an (N, K) polar code.

Unlike the previous two methods, the in-order bit selection method follows the derivation of channel polarization. When evaluating the error probability of bit i, all the former bits from 0 to $i-1$ are already frozen. In this way, the measured error probability of each bit will be reliable, as they cannot be affected by the former frozen low-capacity bits. Therefore, the bit selection using the in-order method is also reliable. The complexity of the in-order bit selection algorithm is $\mathcal{O}(NN_{MC})$.

The in-order bit selection method requires reliable measurement of error probability using Monte Carlo simulations. The number of Monte Carlo simulations, N_{MC}, depends on error rate. The lower the error rate, the more the number of Monte Carlo simulations is required to collect enough errors. In short, the complexity of our method scales inversely with BER. As the code length increases, N increases and BER decreases, which in turn increases N_{MC}. The exact complexity scaling factor depends on how BER decreases with increasing code length. If we hold code length constant and decrease code rate, BER decreases, which in turn increases N_{MC}. The exact complexity scaling factor depends on how BER decreases with decreasing code rate. To speed up in-order bit selection, we use FPGA to accelerate the inner loop (Monte Carlo simulation), and the outer loop is done using a script that interacts with the FPGA accelerator.

The number of Monte Carlo runs N_{MC} can be adjusted for each bit. For a reliable bit, more Monte Carlo runs are required to collect enough errors to obtain a statistically significant error probability measurement, and to differentiate the reliabilities of different bits for ranking and bit selection. On the other hand, for an unreliable bit, the number of Monte Carlo runs can be reduced to save time. Based on this idea, we designed a three-pass scheme. In each pass, we run the in-order bit selection algorithm with a higher N_{MC}. The least reliable bits are identified and excluded in the first pass, followed by the medium reliable bits in the second pass, and the most reliable and hard-to-distinguish bits in the third pass. Designing low rate codes requires more passes as only a small set of the best bits are chosen.

A. Optimal SNR for Bit Selection

We use an AWGN channel in the Monte Carlo simulations. The best bit selection is expected to vary across SNR. To confirm, we first obtain the bit selections using the in-order method at different SNRs, and then test the performance of the bit selections. The FER of the five (256, 128) codes with bit selections done at SNR from 1 dB to 5 dB are shown in Fig. 10. The performance of the five codes vary widely. The codes designed at low SNRs perform worse, especially at moderate to high SNR. The codes designed at high SNRs exhibit much better performance throughout the SNR range.

To understand the implication of SNR on the bit selection, we use TABLE II to show the minimum-distance error distribution for five (256, 128) polar codes with different frozen sets. In this table, minimum-distance errors refer to errors where the Hamming distance from the received codeword to the decoded codeword is smaller than the Hamming distance from the received codeword to the transmitted codeword. Each code's performance is displayed in one column: the bit selections of the five codes are done using the in-order bit selection method at SNR from 1 dB to 5 dB. Not surprisingly, at a high (decoding) SNR, the majority of the errors are due to minimum-distance errors. Therefore if the bit selection is done at a high SNR, the resulting bit selection will specifically reduce the minimum-distance errors and increase the minimum distance of the code.

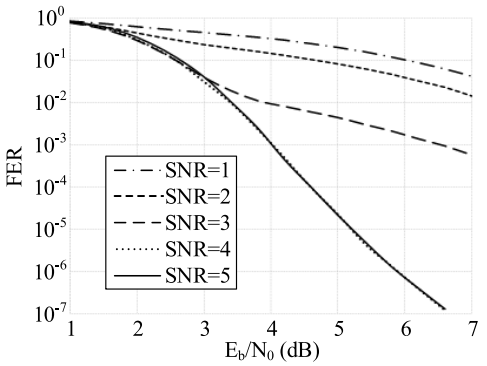


Fig. 10. FER performance of five (256, 128) polar codes designed at SNR from 1 dB to 5 dB.

TABLE II
PERCENTAGE OF MINIMUM-DISTANCE ERRORS OF SIX (256, 128) POLAR CODES DECODED BY BP

		bit selection SNR				
		1 dB	2 dB	3 dB	4 dB	5 dB
decoding SNR	1 dB	30%	20%	0%	0%	0%
	2 dB	75%	50%	5%	0%	0%
	3 dB	95%	95%	20%	0%	0%
	4 dB	100%	95%	85%	0%	0%
	5 dB	100%	100%	100%	0%	0%
	6 dB	100%	100%	100%	0%	0%

Comparing the results presented in TABLE II across the columns in one row, the bit selections done at a high SNR yield fewer minimum-distance errors when simulated in the same channel condition, indicating a larger minimum distance for these codes. Therefore it is expected that a bit selection done at a high SNR has a larger minimum distance.

To run bit selection using the simulation-based in-order selection method, we need the SNR to be sufficiently high, such that the minimum-distance errors dominate the error profile. However, our method is not sensitive to SNR. The optimal design SNR is found to be around 4 dB for an AWGN channel, and it is independent of rate or code length of practical interest, i.e., from 256 to 4K.

B. Implementation and Acceleration of Bit Selection

In our design, part of the bit selection method is implemented on FPGA and part is implemented in software script that interacts with the FPGA. To support the in-order bit selection, we need to measure a bit's error probability. So a N-to-1 MUX, done in a tree structure, is added to the decoder

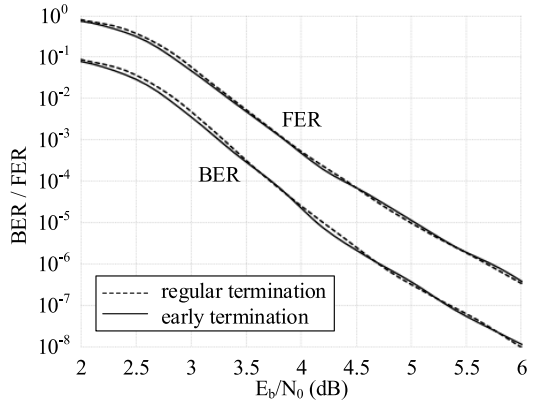


Fig. 11. FER performance of two (1024, 512) polar codes, one designed without early termination and one designed using early termination.

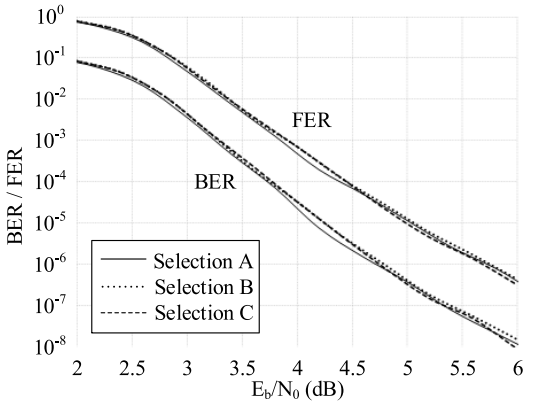


Fig. 12. FER performance of three (1024, 512) polar codes that are designed using different fixed-point quantization schemes.

on FPGA to select one of N bit decisions. To run the bit selection, the software script sets up the the frozen set for the decoder, and selects the appropriate bit to monitor its BER. Once the BER is properly measured, the software script moves to the next bit.

BP decoding is iterative and more iterations tend to improve the error-correcting performance. For the results presented so far, we have used a maximum iteration $L = 15$ to limit the simulation time. To further speed up the bit selection, one effective approach is early termination [22], since the decoding for the majority of the inputs converges after a small number of iterations (much less than 15). However, unlike an LDPC code, there is not a clear convergence indicator for polar codes.

We propose an approximate convergence detection by monitoring the hard decisions for consecutive iterations. If each bit obtains an identical hard decision for T consecutive iterations, the decoder is allowed to terminate. The cost of implementing consecutive decision matching is relatively low, requiring only $(T-1)$ 2-input XNOR gates per bit, and one $N(T-1)$-input AND gate at the top level. To prevent mis-detection, a second criterion is added to ensure that a minimum number of iterations M is met. As Fig. 11 shows, the code designed with early termination ($M = 3$, $T = 3$ and maximum 15 iterations) has similar error-correcting performance as the code designed without early termination (maximum 15 iterations), but early termination can speed up bit selection by up to 5 times.

The decoder used for bit selection can adopt a short word length to reduce its footprint on an FPGA and to reduce the minimum clock period. For example, comparing a 6-bit decoder with an 8-bit decoder implemented in the identical stage-parallel architecture on FPGA, the 8-bit decoder costs 10% more registers, 20% more LUTs, 30% more slices and 25% more RAMs, and the minimum clock period has to be relaxed by 40%. Comparing the error-correcting performance of three bit selections obtained using three different decoders, we see in Fig. 12 that the performance of these different bit selections are similar. In particular, bit selection A is obtained using a Q6.0 (6-bit integer including a sign bit, and 0-bit fraction) decoder; bit selection B is obtained using a Q6.2 decoder; and bit selection C is obtained using a Q8.0 decoder. The three bit selections are simulated using a Q6.0 decoder to obtain the FER/BER curves in Fig. 12. The error-correcting performance of the three constructions being nearly indistinguishable justifies a smaller decoder design to be used to permit a higher degree of parallelism to speed up bit selection.

TABLE III
TIME REQUIRED FOR BIT SELECTION IN SOFTWARE (C SIMULATION ON A MICROPROCESSOR) AND FPGA

	256-bit code			1024-bit code		
SNR	N_{MC}*	μP*	FPGA	N_{MC}	μP	FPGA
1	50k	65 min	33 sec	500k	8.8 day	38 min
2	200k	4.3 hr	2.2 min	2M	35 day	2.5 hr
3	500k	10.8 hr	5.5 min	5M	3 mon	6.3 hr
4	2M	1.8 day	22 min	20M	1 yr	1 day
5	5M	4.5 day	55 min	50M	2.5 yr	2.6 day

*: number of Monte Carlo simulations per bit *: Microprocessor

VI. RESULTS

We used the in-order bit selection method to design three polar codes, a (256, 128) code, a (512, 256) code, and a (1024, 512) code. The FPGA resource utilization is shown in Table I. The FER and BER of the three codes using BP decoding are compared with the bit selection by density evolution for an AWGN channel at 4 dB [18] in Fig. 13. Compared to the codes designed by density evolution for an AWGN channel, our (256, 128) code's BER performance is similar, but our (512, 256) code outperforms at low SNR, and our (1024, 512) code achieves 0.3 to 0.6 dB coding gain over a wide SNR range.

The in-order bit selection for a 256-bit polar code requires on the order of 50k Monte Carlo simulations per bit at 1 dB SNR, or 2M simulations per bit at 4 dB SNR, as more simulations are necessary at a high SNR due to the lower BER. As discussed previously, to obtain a good bit selection, the bit selection needs to be done at a relatively high SNR instead of a low SNR. For a longer code, the simulation time increases further due to the lower BER and more bits in a

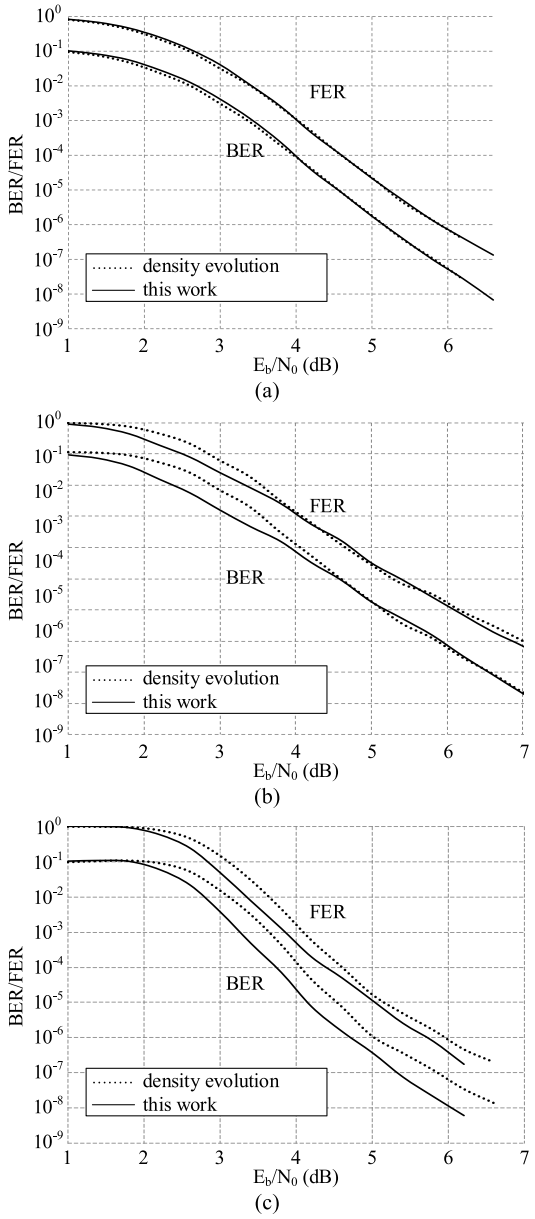


Fig. 13. Performance of (a) (256, 128) (b) (512, 256) (c) (1024, 512) polar codes designed with in-order bit selection.

longer code. The number of simulations per bit for a 1024-bit code is one order of magnitude higher than a 256-bit code.

In TABLE III, we compare the time required for the bit selections using a compiled C code running on an Intel Core i7-4790K processor (quad core, 8M cache, 4.40 GHz, 32GB memory) without multi-threading or SIMD extension and using a Xilinx Virtex-6 SX475T FPGA (100MHz) to provide acceleration. The decoder is implemented in a stage-parallel architecture. The speedup by FPGA is significant: at 4 dB SNR, the 256-bit code selection requires 1.8 days on a microprocessor but only 22 minutes with the FPGA – a 120 times speed up. The estimated C code simulation time for the 1024-bit code is about 1 year, making it impractical. However, with the FPGA, the bit selection can be done in 1 day – a 360 times speedup. The comparison demonstrates the need for acceleration in simulation-based bit selections. One can also use multiple cores and SIMD on CPUs or GPUs to speed up bit selections.

VII. Conclusion

In this work, we present a simulation-based bit selection method and an acceleration methodology to design polar codes for BP decoding. Starting with two hypothetical bit selection methods and the analysis of their weaknesses, we present an in-order bit selection method that bases the frozen set selection on a reliable evaluation of the bit error probability. Nonidealities of the implementation are also accounted for, such as finite block length and fixed-point quantization. The method is applicable to different code rates, code lengths, and channels, and even channels without explicit models. As a result, the method can be deployed in the field to select bits based on the physical channel. The results are demonstrated in three code designs, a (256, 128) code, a (512, 256) code, and a (1024, 512) code. Compared to the codes designed by density evolution for an AWGN channel, our (512, 256) code improves coding gain in BP decoding at low SNR, and our (1024, 512) code improves coding gain in BP decoding by 0.3 to 0.6 dB over a wide SNR range.

To speed up the simulation-based bit selection, we make use of a stage-parallel BP decoder on FPGA. The inner loop of the bit selection is done on FPGA to cut the simulation time by orders of magnitude. To further speed up the bit selection, we implement an early termination scheme to shorten the decoding latency by up to 5 times. As a result, the bit selection for a 256-bit polar code takes only 22 minutes and a 1024-bit polar code takes 1 day, making it feasible for designing codes of practical block lengths.

Acknowledgment

The authors would like to thank Y. Tao and Dr. Y. S. Park for establishing the initial BP decoder simulation, Prof. W. J. Gross for providing the density evolution bit selection, and Dr. F. Sheikh and collaborators at Intel Labs for supporting this research and providing advice.

References

[1] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[2] B. Zhang *et al.*, "A 5G trial of polar code," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2016, pp. 1–6.

[3] D. Zhang. (2016). *Up in the Air With 5G*. [Online]. Available: http://www.huawei.com/en/publications/communicate/80/up-in-the-air-with-5g

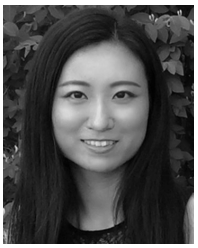
[4] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2011, pp. 1665–1668.

[5] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Trans. Signal Process.*, vol. 61, no. 10, pp. 2429–2441, May 2013.

[6] A. Pamuk, "An FPGA implementation architecture for decoding of polar codes," in *Proc. Int. Symp. Wireless Commun. Syst.*, Nov. 2011, pp. 437–441.

[7] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68 Gb/s belief propagation polar decoder with bit-splitting register file," in *Symp. VLSI Circuits Dig. Tech. Papers*, Jun. 2014, pp. 1–2.

[8] B. Yuan and K. K. Parhi, "Architecture optimizations for BP polar decoders," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 2654–2658.
[9] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.
[10] U. U. Fayyaz and J. R. Barry, "Low-complexity soft-output decoding of polar codes," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 958–966, May 2014.
[11] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, Oct. 2012.
[12] Y. Wang and K. R. Narayanan, "Concatenations of polar codes with outer BCH codes and convolutional codes," in *Proc. 52nd Annu. Allerton Conf. Commun., Control, Comput.*, Sep. 2014, pp. 813–819.
[13] J. Guo, M. Qin, A. G. I. Fabregas, and P. H. Siegel., "Enhanced belief propagation decoding of polar codes through concatenation," in *Proc. IEEE Int. Symp. Inform. Theory*, Jun. 2014, pp. 2987–2991.
[14] C. Zhang, X. Yu, and J. Sha, "Hardware architecture for list successive cancellation polar decoder," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 209–212.
[15] J. Lin, C. Xiong, and Z. Yan, "Reduced complexity belief propagation decoders for polar codes," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2015, pp. 1–6.
[16] R. Mori and T. Tanaka, "Performance and construction of polar codes on symmetric binary-input memoryless channels," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2009, pp. 1496–1500.
[17] R. Mori and T. Tanaka, "Performance of polar codes with the construction using density evolution," *IEEE Commun. Lett.*, vol. 13, no. 7, pp. 519–521, Jul. 2009.
[18] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.
[19] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Trans. Commun.*, vol. 60, no. 11, pp. 3221–3227, Nov. 2012.
[20] C. Xiong, Y. Zhong, C. Zhang, and Z. Yan, "An FPGA emulation platform for polar codes," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2016, pp. 148–153.
[21] J. Wuthrich, A. Balatsoukas-Stimming, and A. Burg, "An FPGA-based accelerator for rapid simulation of SC decoding of polar codes," in *Proc. IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2015, pp. 633–636.
[22] B. Yuan and K. K. Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE Trans. Signal Process.*, vol. 62, no. 24, pp. 6496–6506, Dec. 2014.



Shuanghong Sun (S'11) received the B.S. degree in electrical and computer engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012, and the B.S. and M.S. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2012 and 2014, respectively, where she is currently pursuing the Ph.D. degree in electrical engineering.

She was with Broadcom Ltd., Irvine, CA, and Qualcomm Inc., San Diego, CA, in 2015. Her research interests are channel coding, digital architectures, and high-performance VLSI systems.

Zhengya Zhang (S'02–M'09) received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley (UC Berkeley), Berkeley, CA, USA, in 2005 and 2009, respectively. He has been with the faculty of the University of Michigan, Ann Arbor, since 2009, where he is currently an Associate Professor with the Department of Electrical Engineering and Computer Science. His current research interests include low-power and high-performance VLSI circuits and systems for computing, communications, and signal processing.

Dr. Zhang was a recipient of the National Science Foundation CAREER Award in 2011, the Intel Early Career Faculty Award in 2013, the David J. Sakrison Memorial Prize for Outstanding Doctoral Research in electrical engineering and computer sciences at UC Berkeley, and the Best Student Paper Award at the Symposium on VLSI Circuits. He was a past Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—Part I: Regular Papers from 2013 to 2015 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—Part II: Express Briefs from 2014 to 2015. He has been serving as an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS since 2015.